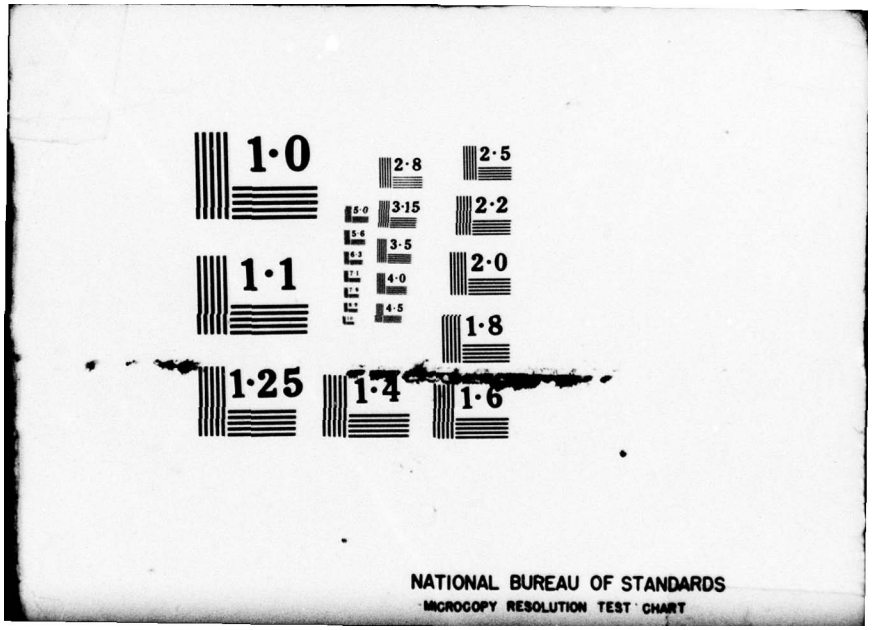
AD-A058 565 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MASS C--ETC F/G 9/2
AN EXTENDED MODEL FOR A SYSTEMATIC APPROACH TO THE DESIGN OF CO--ETC(U)
JUL 78 S L HUFF, S E MADNICK N00039-78-G-0160
UNCLASSIFIED CISR-P010-7806-07 NL

1 OF 1
ADA
058565

END
DATE FILMED
11-78
DDC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART


12

LEVEL II



Center for Information Systems Research
Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139
617 253-1000

(15) Contract Number N00039-78-G-0160

Internal Report Number P010-7806-07

Deliverable Number 003

(12)

LEVEL II

AD A058565

AD No. _____
DDC FILE COPY

(14) CISR-P010-7806-07.
CISR-TR-7

(6) An Extended Model
for a
Systematic Approach
to the
Design of Complex Systems.

(9) Technical Report

(12) 67p.

(10) S. L. Huff
S. E. Madnick

(11) July 1978



Principal Investigator:

Prof. S. E. Madnick



Prepared for:

Naval Electronic Systems Command
Washington, D.C.

409 590

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>Technical Report #7 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>"An Extended Model for a Systematic Approach to the Design of Complex Systems" | | 5. TYPE OF REPORT & PERIOD COVERED |
| | | 6. PERFORMING ORG. REPORT NUMBER<br>P010-7806-07 |
| 7. AUTHOR(s)<br>S. L. Huff<br>Prof. S. E. Madnick | | 8. CONTRACT OR GRANT NUMBER(s)<br>N00039-78 G-0160 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Center for Information Systems Research<br>Sloan School of Management, M.I.T.<br>Cambridge, Mass. 02139 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Naval Electronic Systems Command | | 12. REPORT DATE<br>July 1978 |
| | | 13. NUMBER OF PAGES<br>63 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>Unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software requirements analysis; functional requirements specification; software architectural design; problem design structuring; mathematical graph modelling.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The objective of this study is to develop a systematic approach to the architectural design of complex software systems. This contract builds on earlier work, in which a graph modelling and decomposition methodology was used to operate upon a set of functional requirements and their interrelationships to generate an architectural design. In this report, certain extensions to the graph model employed to model the requirements are analyzed. Proposed extensions include: (a) implementation nodes; (b) weights on inter-

→ next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

conti

dependency links; (c) links between implementation nodes; and (d) various types of directed links. The proposed extensions are applied to a small design problem (the design of a 22-requirement database management system) used in earlier work, and found to be implementable - that is, the information that must be supplied by a software designer to establish the model structure in a particular case can be determined in a reasonable length of time.

## PREFACE

The Center for Information Systems Research (CISR) is a research center of the M.I.T. Sloan School of Management. It consists of a group of management information systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, implementing, and maintaining application software, information systems, and decision support systems.

Within the context of the research effort sponsored by the Naval Electronics Systems Command under contract N00039-78-G-0160, CISR has proposed to conduct basic research on a systematic approach to the early phases of complex systems design. The main goal of this work is the development of a well-defined methodology to fill the gap between system requirements specification and detailed system design.

The research being performed under this contract builds directly upon results stemming from previous research carried out under contract N00039-77-C-0255. The main results of that work include a basic scheme for modelling a set of design problem requirements, techniques for decomposing the requirements set to form a design structure, and guidelines for using the methodology developed from experience gained in testing it on a specific, realistic design problem.

The present study aims to extend and enhance the previous work, primarily through efforts in the following areas:

1) additional testing of both the basic methodology, and proposed extensions, through application to other realistic design problems;

2) investigation of alternative methods for effectively coupling this methodology together with the preceding and following activities in the systems analysis and design cycle;

3) extensions of the earlier representational scheme to allow modelling of additional design-relevant information;

4) development of appropriate graph decomposition techniques and software support tools for testing out the proposed extensions.

This document, which relates primarily to category (3) above, includes an investigation of various possible extensions to the network model used to represent the system requirements and their interrelationships. These extensions are assessed in terms of the additional design-relevant information they allow a system designer to express. A small example design problem is analyzed using the proposed extensions.

## EXECUTIVE SUMMARY

Complex design problems are characterized by a multitude of competing requirements. System designers frequently find the scope of the problem beyond their conceptual abilities, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable sub-problems. Functional requirements form a key interface between the users of a system and its designers. In this research effort, a systematic approach has been proposed for the decomposition of the overall set of functional requirements into sub-problems to form a design structure that will exhibit the key characteristics of good design: strong coupling within sub-problems, and weak coupling between them.

In this report, certain extensions to the graph model employed to model the requirements that make up the problem specification, are analyzed. The graph model used previously in this research was constrained in that it only included one type of node (corresponding to system functional requirements) and one type of link (binary links, representing implementation interrelationships).

A number of extensions are proposed, including:

- weights on interdependency links, to represent interdependency strength;

- an additional node type - the implementation node - to represent implementation issues explicitly;

- links between implementation nodes, to represent commonalities among implementation issues;

- directed links between requirements, to represent implication relationships;

- directed links between requirements, to represent hierarchical relationships;

- directed links between implementation nodes, to represent implication relationships between implementation issues.

The proposed extensions are investigated further in the context of a small design problem, a 22-node database management system (DBMS) specification. The extensions are all found to be implementable. That is, the types of questions that a designer would have to answer (to his own satisfaction) in order to elicit the information necessary for constructing the extended graph model:

(1) could in fact be answered satisfactorily, and

(2) could be addressed in a reasonable length of time.

In this report, the relative value of each of the various extensions is not directly addressed, pending further experience with application of the methodology. Neither is the extended graph model of the 22-node DBMS specification analyzed further. Such analysis (graph decomposition, interpretation, etc.) requires some new tools currently under development, and will be presented in a future report.

## TABLE OF CONTENTS

## 1. Introduction.

The problem of designing quality software systems has existed practically as long as computers themselves. Only recently, however, have efforts been made to develop techniques to aid software designers in their jobs - in effect, attempting to add an element of science to the software design craft. One such effort is the Systematic Design Methodology (SDM), a set of concepts and techniques currently under development at the Center for Information Systems Research (CISR), at the Sloan School of Management, M.I.T. The SDM is oriented toward assisting software designers (or design teams) in the task of structuring the architecture - the preliminary design - for a complex system (Madnick and Andreu, 1977).

Underlying the SDM approach is a technique for modelling the design problem by representing a system's functional requirements and their interdependencies as an undirected graph with unweighted (binary) links. This basic model and methodology have been applied to some experimental systems ((Andreu, 78), (Holden, 78)), and have been found to be an effective means of determining an initial design problem structure.

The purpose of this study is to examine the representational scheme used within the Systematic Design Methodology, and to suggest certain extensions to enhance the modelling power of this scheme. The SDM basic model, together with the extensions discussed in this report, will be referred to as the "extended model."

The extended model is applied to a simple design problem,

featuring 22 requirements for the design of a database management system. This system was also studied earlier by Andreu (Andreu, 77). Comparisons with Andreu's representation are drawn. Later reports will further this investigation by examining graph partitioning methods that might be used with the extended model, and measures to reflect the goodness of such a partition.

## 2. Overview of the Basic Model.

The design structuring methodology reported in (Andreu, 78) forms a basis for the present work. At the core of this methodology is a simple model (the "basic model") used to represent general design structuring problems.

The basic model is an undirected graph: a set of nodes, together with a set of unweighted connecting links. Each functional requirement in the system specification is represented as a separate node(1). A link joining two nodes corresponds to an interdependency between these nodes. Ways in which both nodes and links are determined are discussed below.

### 2.1 Generation of Nodes in the Basic Model

Each node represents a single functional requirement of the target system. Desirable properties of these functional requirement statements include:

(a) unifunctionality - each statement describes a single function (not multiple functions) to be featured in the target system;

(b) implementation independence - each statement should be implementation free, i.e., ought to specify what is required of the target system but not how that requirement is to be met.;

(c) common conceptual level - all requirement statements should be, to the extent possible, at the same level of generality, or abstraction.

In Andreu's research, the problem of creating the functional

---

(1) Functional requirement statements must be in an appropriate form. This issue is discussed in (Andreu, 78) and further in (Huff, 78).

requirements for a system has not been directly addressed; instead, they were taken as given(2). One scheme for mapping English-language prose requirement specifications into an appropriate set of functional requirement statements according to the above guidelines is discussed in (Huff 78). Additional work on this problem is planned.

For the purposes of this report, the assumption will again be made that appropriate functional requirement statements are given to the system designer.

## 2.2 Generation of Links in the Basic Model.

Interdependencies between pairs of requirements are represented as links, or "edges", in the basic graph model. Andreu and others have discussed in some detail the interpretation of design interdependencies. For example, Andreu writes:

> "In essence, two requirements are interdependent when one can think of plausible implementation schemes in which the two ought to be considered simultaneously for design purposes, the reason being that if such an implementation scheme was to be adopted, meeting these requirements would be one of the central considerations that the designer should take into account to tailor the scheme to the characteristics of the design in progress."(3)

Furthermore, interdependencies fall naturally into two

---

(2) Andreu did discuss guidelines for inspecting and verifying the requirement statements.

(3) see (Andreu, 78, page 70).

groups, termed concordant and discordant.(4) A concordant interdependency exists between requirements A and B if the implementation of requirement A would tend to simplify, assist, or otherwise make easier the implementation of requirement B. In contrast, a discordant interdependency exists between the two requirements if implementation of A would hamper, jeapordize, or otherwise make more difficult the implementation of B.

The basic approach suggested by Andreu for actually determining the interdependencies has the designer consider each requirement pair in turn, and mentally consider the alternative approaches he might follow in implementing the two requirements. The various implementation schemes so conceived are termed "mental models" of implementation. If the designer perceives a significant degree of interaction, in the context of his mental models, between a given requirement pair, he interprets the requirement pair as being interdependent.

The actual determination of interdependencies and their nature (concordant or discordant) is heavily designer-dependent. There is no intent within the methodology to remove or de-emphasize the designer's experience or judgment from the design activity. Rather, the methodology attempts to provide structure and simplification to the decisions the system designer must make. By only having to consider a pair of requirements at one time, rather than the entire set, the cognitive demands on the designer are greatly reduced. There is a price to be paid

---

(4) Andreu used the terms "concurrency" and "tradeoff".

for this simplification, however: now the designer has easier decisions to make (pairwise comparisons), but more of them. While not as formidable a task as might appear at first glance, the complete assessment of interdependencies for a non-trivial problem involves considerable effort.

## 2.3 An Example.

To illustrate more concretely the ideas discussed in this paper, a specific real (but small) design problem will be studied in terms of both the basic model and the extended model. The problem concerns the design of a database management system (DBMS). A set of 22 functional requirements is assumed given (refer to Section 4.1 for a listing of these requirements).

Requirement interdependencies were assessed for this requirement set in the fashion described in the previous section. A total of 38 interdependencies were determined. Descriptions of each interdependency are given in Section 4.2 of this report, where the example system is examined in more detail.

These requirements and interdependencies may be displayed as a graph, following the basic model, as shown in Figure 2.1. Of course, determining the graph structure for the system requirements is only the first step in the basic design methodology. Further steps, including partitioning the graph appropriately, and interpreting design subproblems and their interactions, would normally follow. As this report is only concerned with representational issues, further steps such as these will be analyzed in later reports.

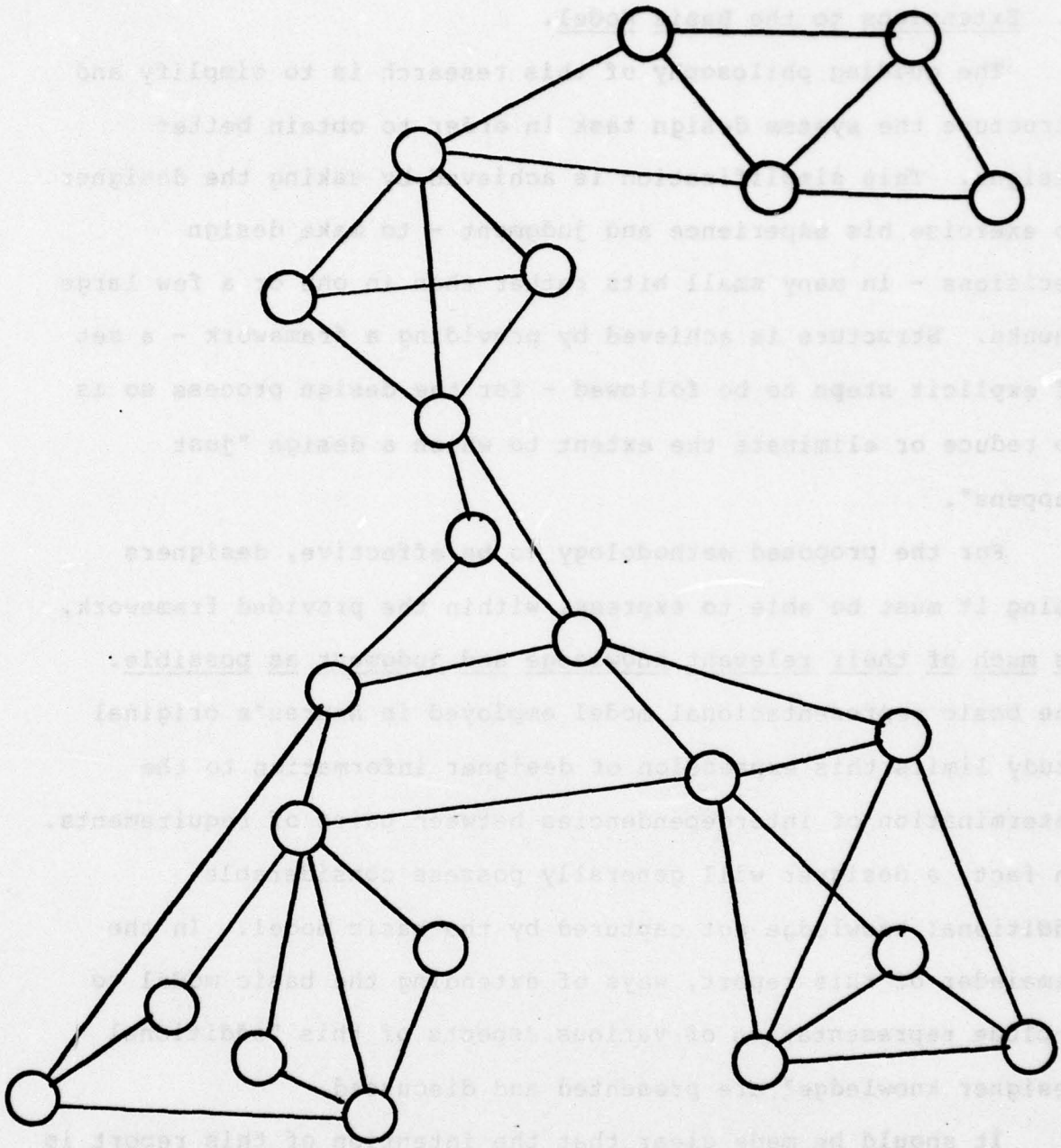


**Figure 2.1**

**Graph representation of 22-node DBMS design problem**

## 3. Extensions to the Basic Model.

The guiding philosophy of this research is to simplify and structure the system design task in order to obtain better designs. This simplification is achieved by asking the designer to exercise his experience and judgment - to make design decisions - in many small bits rather than in one or a few large chunks. Structure is achieved by providing a framework - a set of explicit steps to be followed - for the design process so as to reduce or eliminate the extent to which a design "just happens".

For the proposed methodology to be effective, designers using it must be able to express, within the provided framework, as much of their relevant knowledge and judgment as possible. The basic representational model employed in Andreu's original study limits this expression of designer information to the determination of interdependencies between pairs of requirements. In fact, a designer will generally possess considerable additional knowledge not captured by the basic model. In the remainder of this report, ways of extending the basic model to include representation of various aspects of this "additional designer knowledge" are presented and discussed.

It should be made clear that the intention of this report is to lay out and analyze various possible kinds of extensions that could be made to the basic model. No choices will be made at this time as to which, if any, of these extensions will in fact be adopted for further research in this problem area. However, in conjunction with the example discussed in Section 4, the ease

of application of each extension is examined briefly, and tentative conclusions about priorities regarding extensions to the basic model are drawn.

### 3.1 Interdependency Weights.

In the basic model, an interdependency either exists or it doesn't - there is no middle ground, no notion of the "strength" of an interdependency. Links within the graph representation of a design problem are binary in nature: the adjacency matrix is a matrix of ones and zeros.

There is nothing inherent in the design problem representation that necessitates binary links, other than a desire to work with as simple and parsimonious a model as possible. On the other hand, there is good reason to relax this requirement, namely, that important aspects of designer knowledge might thereby be included in the design problem model.

One possible extension of the basic model would be the association of a "weight" W(i,j) with each link. The inclusion of such link weights in the graph model can be likened to course grading: binary links would correspond to pass-fail grading, whereas weighted links correspond to standard (letter or numerical) grading. Just as standard grading allows an instructor to express more information about his or her students, so weighted links would serve to capture more of the designer's judgment concerning the relationships among system requirements.

There is a variety of possible ways in which such a weight could be defined and justified. For example, the weight W(i,j)

could be taken to represent the "closeness", or "strength of interdependence" of the requirements i and j in the context of the interdependency represented by link L(i,j). With this interpretation, two requirements that are seen to be closely related, in implementation terms, would be connected by a link with a relatively high assigned weight.

Alternatively, link weights could be defined in a subjective probability sense. In this case, the weight W(i,j) would represent the degree of uncertainty in the designer's mind that the requirements i and j will interact in implementation. This interpretation would be consonant with the uncertainty inherent in the interdependency assessment process itself. With this definition, a designer who is quite certain that two requirements i and j would be coupled in implementation would assign a relatively high weight (close to 1) to the link L(i,j); whereas if the designer believes that there is only a fairly small liklihood of the requirements being coupled, the assigned weight would be lower.

These alternative definitions of link weight give rise to four logical combinations, as displayed below.

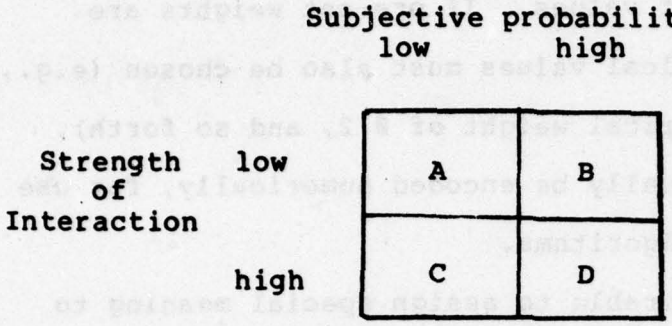



Figure 3.1

In fact, these definitions are not completely "orthogonal". If, for example, two requirements are seen to be strongly related, according to the first definition, then there will be a tendancy for designers to view the probability of their being related as high. Thus, it is reasonable to assume that most designers' weight assignments would fall in cells A and D in the above diagram.

### 3.1.1 Scaling Problems.

A decision to include link weights in the graph model gives rise to certain scaling problems. First, a range must be specified over which weights will be allowed to vary. While not absolutely necessary, compatability with the basic model, among other reasons, suggests that link weights be chosen from the numerical range [0,1] .

A choice must also be made between requiring the designer to choose from "pre-set" weight values (e.g., low, medium, high)

versus a continuous range of values. If pre-set weights are used, some mapping to numerical values must also be chosen (e.g., "low" corresponds to a numerical weight of 0.2, and so forth). All link weights must eventually be encoded numerically, for use in the graph partitioning algorithms.

Also, it may prove desirable to assign special meaning to certain weight values. For instance, a special "very high" weight category might be defined to allow a designer to express his conviction that two requirements absolutely must be included in the same sub-problem.

In contrast, a "very low" weight may be used to specify that two requirements must be in different sub-problems. However, there are also certain arguments for avoiding such deterministic weight assignments, in that they reduce design flexibility are partially preempt the design structuring methodology itself.

For the purposes of implementing extensions to the basic model, hard choices need not always be made among alternative representation issues ahead of time. A software package that designers would use to apply these architectural design methods could allow the user to select, from various options, those alternatives that most appealed to him, that he found easiest to use, etc.

Our experiences to date indicate that system designers most likely would not require the capability of specifying weights directly in numerical terms. The following scheme, using a

simple three-way breakdown, has proven effective and easy to use:

| Designer's Judgment About Weight | Code | Numerical Value |
|---|---|---|
| Strong | S | 0.8 |
| Average | A | 0.5 |
| Weak | W | 0.2 |

This particular encoding of weights achieves a number of useful results. First, the three basic weightings may be easily interpolated, as shown below:

| Weight: | S+ | S | S- | A+ | A | A- | W+ | W | W- |
|---|---|---|---|---|---|---|---|---|---|
| Numerical value: | 0.9 | 0.8 | 0.7 | 0.6 | 0.5 | 0.4 | 0.3 | 0.2 | 0.1 |

Secondly, all weights fall in the [0,1] range, a fact useful for normalization and later decomposition purposes. Finally, the end points in the range (the values 0 and 1) may be reserved for "must be included" and "must not be included" categories, if so desired.

There is another, more subtle, set of scaling issues that involves the correctness, or accuracy, of link weight assignments. If, in assessing a set of requirements, a designer assigns a weight of, say, 0.85 to a certain link, how does he know that it should be 0.85, and not 0.84 or 0.86? Of course, he doesn't know for sure; he is simply trying to quantify as best he can what is inherently a ill-defined issue. This accuracy problem did not first arise with the introduction of link weights into the model. It was present in the basic model as well, in the

guise of deciding whether particular links should or should not be included at all.

The original motivation for augmenting the basic model through the inclusion of link weights (as well as other extensions to be discussed shortly) is that designers could provide additional information, relevant to the design structuring problem, that is not captured by simple binary links. The fact that this information is not one hundred percent "accurate" should not prevent it from being used.(1)

## 3.2 Information Linking Implementation Issues.

In the basic model, two nodes (requirements) are joined by a link when they are deemed to be "implementation interdependent" by the designer. In essence, links represent implementation considerations. In this view of requirements and implementation considerations, the focus is upon the nodes, with links being a kind of implementation "glue" which binds the nodes together.

A different way of viewing links is as "things", or logical entities, in their own right, rather than just bindings. To some extent, the appropriateness of viewing an implementation issue as a logical entity depends on the specificity of the statement describing that issue. Not all interdependencies can be specified by a designer with the same level of precision. In

---

(1) There is a strong similarity between this argument and that put forth by Bayesian statisticians in defending the use of personal, or subjective, probabilities in statistical models.

some cases, it is possible for designers to indicate in some detail the nature of the interaction that he believes will occur in the process of implementing a given pair of requirements. Requirements for "fast direct-access retrieval response" and for "ability to perform sequential file processing efficiently" may be assessed as concordant requirements since, the designer may believe, the use of the Indexed Sequential Access Method (ISAM) file organization is required in both cases. In such a situation, the link joining this pair of requirements represents more than just general implementation interdependence; in particular, it represents the use of ISAM file organization, a rather specific implementation scheme.

In other cases, a designer will not be able (at this stage in the design process) to be so specific. He may believe that a pair of requirements will prove to be discordant (interfere with each other) as the design proceeds, but may not, as yet, be able to specify how this interference will come about.

It is useful to treat interdependencies as separately identifiable issues for another reason also, namely, for documenting project design decisions. It often happens in design and development projects (especially large projects) that early design decisions are made by one person or group, and left undocumented. The responsible individuals may leave the project, or otherwise become "disassociated" from that aspect of the system. Later designers and implementors often come to question what the motivation might have been underlying earlier decisions. Having a well-documented "track" for each design decision would

be quite valuable for determining how later design issues or system modifications ought to be implemented.

Now, viewing links as representing logical entities leads to certain questions about such entities: can these entities themselves be related in ways meaningful to designers and to the design structuring problem? The answer is yes, and the incorporation of such link relationships into the basic model provides additional useful extensions.

There are two different kinds of relationships between implementation issues discussed in this report. In this section, similarity relationships are addressed; implication relationships are addressed later.

### 3.2.1 Similarity Links Among Implementation Issues.

Two or more links may represent the same, or closely related, implementation issues. A simple example of this possibility is illustrated in Figure 3.2. The links joining requirements 1 and 2, and requirements 2 and 3, both represent the interdependency "ISAM organization", an implementation consideration through which both requirement pairs (1,2) and (2,3) are deemed by the designer to be interdependent.

In this example, the two links represent entirely the same implementation issue. In general, however, the degree of "sameness" between two or more implementation issues will usually be less than 100 percent in the eyes of the designer, due to inherent fuzziness in the specification of both functional requirements and implementation schemes. The judgment as to

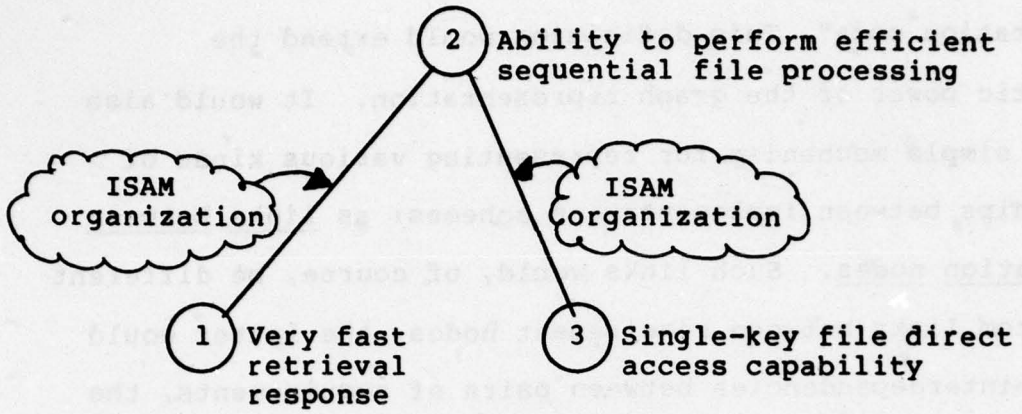



Figure 3.2

whether a given pair of links "really" represent the same implementation issue is, again, a designer decision.

Going one step further, a weight factor could be associated with the similarity assessment to represent the extent to which the designer judges the two implementation issues to be the same. That is, such a weight would correspond to the extent of overlap between the implementation issues, in the designer's estimation.

### 3.2.2 Graph Representation of Implementation Issue Commonality.

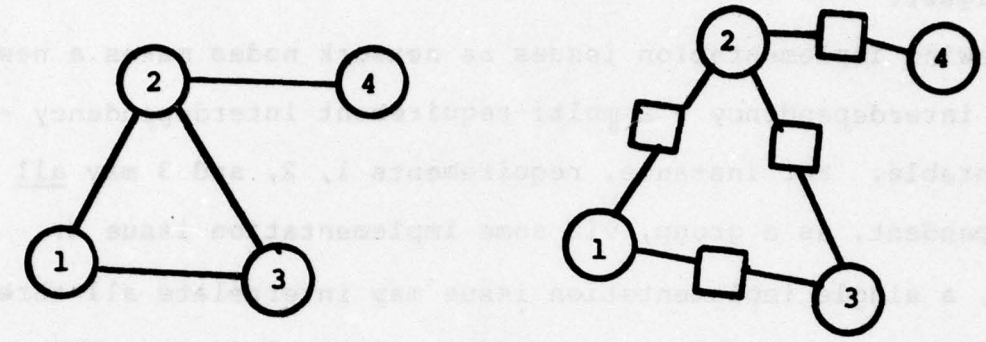
Probably the easiest approach to the schematic representation question is to avoid it, by not actually incorporating implementation similarity information into the graph at all. Rather, the associated information could be kept in a table or matrix, to be included in the model but not actually depicted schematically.

An alternative approach, which would lend itself to easy display, would be to define a new type of node - an

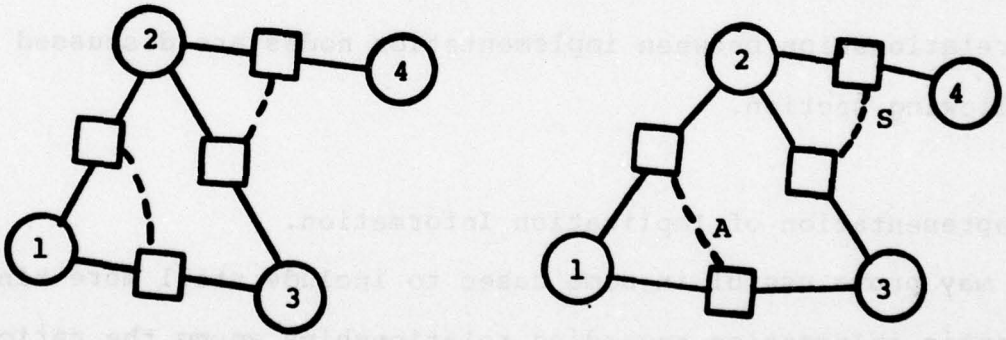
"implementation node". This definition would extend the diagrammatic power of the graph representation. It would also provide a simple mechanism for representing various kinds of relationships between implementation schemes: as links between implementation nodes. Such links would, of course, be different in kind from links between requirement nodes; the latter would represent interdependencies between pairs of requirements, the former would represent relationships between the implementation schemes themselves.

As a diagrammatic technique, requirement nodes (R-nodes, henceforth) ought to be distinguished from implementation nodes (I-nodes). One simple approach would be to use circles for the former (as in the basic model) and, say, squares for I-nodes. Similarity relationships between implementation issues may then be represented by undirected links between the corresponding I-nodes. Furthermore, the degree of similarity could be represented as a link weight, similar to the "strength of interdependence" weights discussed in Section 2.

These diagrammatic ideas are illustrated in Figure 3.3. Figure 3.3(a) shows a simple four-node graph using the style of the basic model. In Figure 3.3(b), implementation nodes are formally indicated and labelled (the specific implementation concept that each such node is intended to represent would presumably be described in accompanying documentation). In Figure 3.3(c), the fact that certain of the I-nodes are judged to represent similar implementation schemes is depicted using links connecting those I-nodes. Finally, in Figure 3.3(d), weights are




(a) (b)




(c) (d)

Figure 3.3

attached to these links to describe the extent of overlap, or similarity, between the implementation issues, as perceived by the designer.

Viewing implementation issues as network nodes makes a new kind of interdependency - a multi-requirement interdependency - representable. For instance, requirements 1, 2, and 3 may all be interdependent, as a group, via some implementation issue X. That is, a single implementation issue may interrelate all three requirements at once, as illustrated in Figure 3.4 (a). This kind of relationship would be different from a trio of similar pairwise interdependencies (see Figure 3.4 (b)), as it would more strongly suggest the appropriateness of clustering the three requirements into a common design sub-problem.

Additional kinds of information that may be represented using relationships between implementation nodes are discussed in the following section.

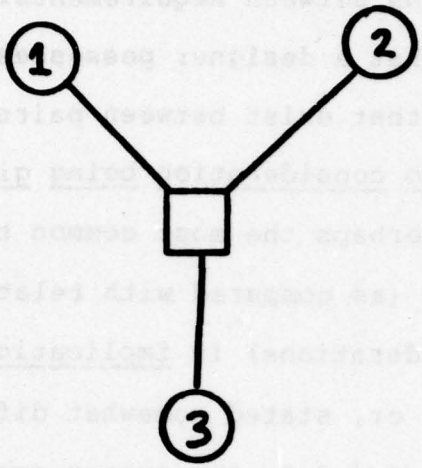
## 3.3 Representation of Implication Information.

It may prove useful in some cases to include still more kinds of semantic information regarding relationships among the various requirements and implementation schemes. Additional considerations not captured in the model extensions discussed so far include:

a) logical implications between requirements at the same level of abstraction, and between requirements at different levels of abstraction;

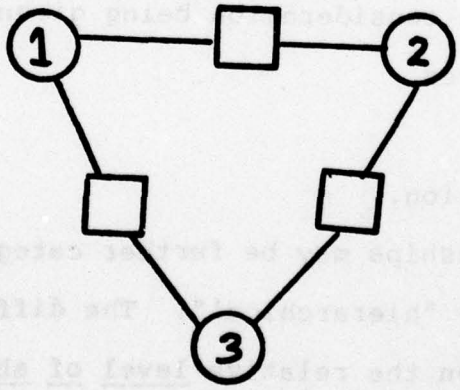
b) logical implications between implementation concepts;

c) logical relationships between requirements and




(a)

A multi-requirement interdependency




(b)

A trio of similar pairwise interdependencies

Figure 3.4

implementation schemes.

### 3.3.1 Logical Implications Between Requirements.

It may be the case that a designer possesses information concerning relationships that exist between pairs of requirements, even with no consideration being given to implementation issues. Perhaps the most common type of such a "functional" relationship (as compared with relationships derived from implementation considerations) is implication: "requirement 1 implies requirement 2"; or, stated somewhat differently, "if requirement 1 is to be included in the system specification, then it follows that requirement 2 must be included also."

It is important to note that such a functional relationship, as defined here, may exist independently of how requirements 1 and 2 are to be implemented. The logic of the relationship is contained in the semantics of the functional requirement statements alone, without consideration being given to implementation alternatives.

#### 3.3.1.1 Lateral Implication.

Implication relationships may be further categorized as being either "lateral" or "hierarchical". The difference between these two types depends on the relative level of abstraction (2)

---

(2) The concept "level of abstraction" is widely used in the system specification and system analysis literature. It is, however, rarely defined, and is generally taken as a kind of primitive concept.

of the two requirements, as seen by the designer. Implication relationships between requirements that exist at the same level of abstraction are laterally related, while those between requirements at different level are hierarchically related.

To make the above ideas clearer, some examples are developed below.

3.3.1.2 Examples.

As an example of logical implications between requirements, consider the pair of requirements

- "capability for collecting resource usage statistics for each submitted job", and
- "able to charge users by department and group for resources used".

These functional requirements are seen to be logically related even without considering implementation alternatives: fulfilling the second requirement implies fulfilling the first. Unless a system is able to account for resource usage, it will have no rational mechanism for meeting the second requirement. Furthermore, the designer would probably view these requirements as existing at a common level of abstraction - i.e., neither is logically contained within the other.

In passing, it might be noted that these requirements are also interrelated at the implementation level, since the implementation of the resource statistics collection subsystem will undoubtedly be affected by considerations of the lines along which charges are to be collected and distributed. It is conceivable, although unlikely, that two requirements could be

related through logical implication at the functional level, yet not be related at the implementation level.

As well as one-way implication, each requirement of a pair could logically imply the other. A possible example of such a pair might be the requirements

- "system must be accessible in conversational mode", and
- "system must be convenient to use".

The designer may feel that an on-line system should always be more convenient to use than a batch system; contrariwise, in order for a system to be truly more convenient to use, it must provide an interactive user interface - i.e., it must be an on-line system.

Diagrammatically, these logical implications between requirement nodes could be represented with a conventional directed link, thus:


(a) One-way (b) Bi-directional

Figure 3.5

An arrowhead is used to distinguish implication relationships

from implementation-level relationships.

3.3.1.3 Hierarchical Implication.

Requirements may be related in an implication sense and also exist at different (hierarchical) abstraction levels. For example, a requirement A may be logically part of another requirement B.

Consider, for instance, the pair of requirements:

- "capability for report formatting", and
- "report formatting optionally automatic."

These requirements are arguably related in a functional sense, but not in the earlier sense of one implying the other at the same level of abstraction. Rather, the first requirement is logically "greater than" the second; the second does not make sense without the first.

It is not necessarily easy to distinguish hierarchical implications from lateral implications. Nevertheless, the differences do seem to be distinguishable in some cases. Since these two types of relationships may be seen to carry different kinds and amounts of design structuring information, it is appropriate to make a differentiation for the purposes of this report.

Hierarchical implication relationships could be represented schematically as shown below, the double line being used to distinguish from both lateral implications and from implementation relationships.

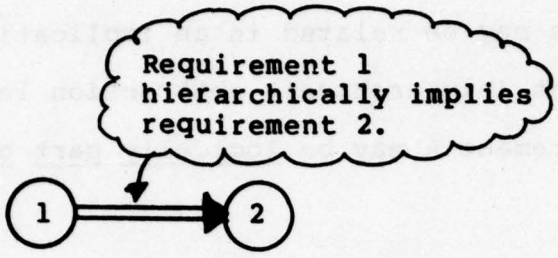



Figure 3.6

3.3.2 Logical Implications Between Implementation Issues.

An argument has been made for the representation of similarity information joining pairs of implementation nodes (I-nodes). Other kinds of relationships between I-nodes may also be represented in the extended model.

The previous arguments regarding logical implication between pairs of I-nodes may be carried across to implementation nodes also. That is, the assumed existence of some implementation scheme X may be viewed by the designer as implying the inclusion of some other scheme, Y.

While it may even be possible to extend the distinction between lateral and hierarchical relationship types to this case, for practical purposes the designer generally does not have a clear enough picture of ultimate detailed implementation techniques during architectural design activity to make such relatively fine distinctions. Therefore, only one "general-purpose" type of logical implication relationship

between I-nodes is considered for inclusion in an extended graph model.

As an example of such a relationship between I-nodes, consider the following three requirements:

1: "Very rapid record retrieval";

2: "Ability to produce sequential listing efficiently";

3: "Ability to access records based on value of key field".

Now, a designer might assess requirements 2 and 3 as being related via an implementation issue X: "ISAM file organization". Similarly, 1 and 3 may be seen to be related via Y: "In-core index tables". Finally, for the sake of illustration, the designer may believe that implementation issue X logically implies implementation issue Y; that is, the use of indexed sequential file organization as an implementation technique implies the use of in-core tables for record lookup.

Borrowing again from the earlier arguments regarding implications among requirements, a straightforward way of portraying I-node implications would be to make use of a directed link joining two implementation nodes. The example discussed above is so illustrated in Figure 3.7.

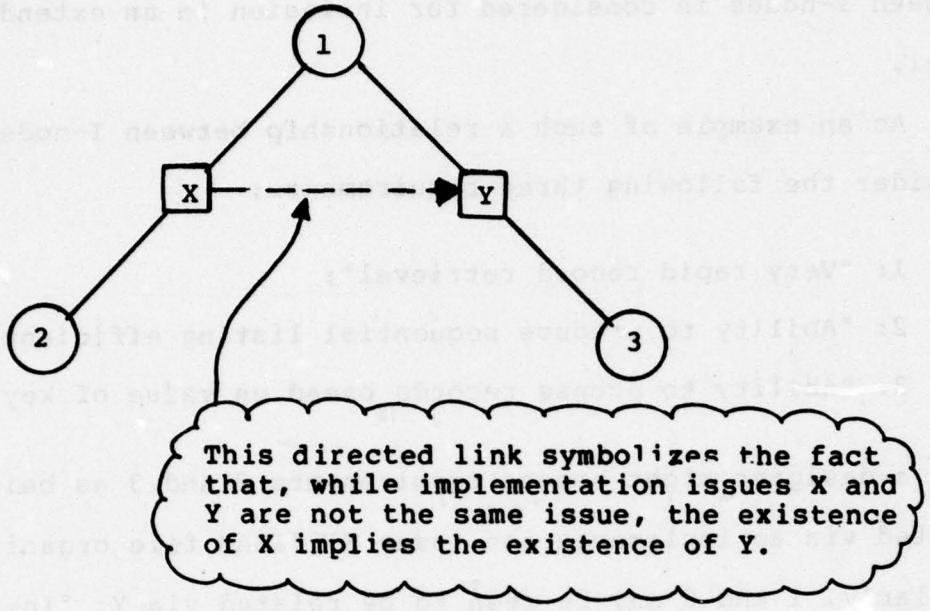



Figure 3.7

3.3.3 Logical Implications between R-nodes and I-nodes.

In the previous section, arguments have been given for including in the extended model various means for representing logical implication relationships between R-node pairs, and between I-node pairs. For completeness' sake, consideration should also be given to implications between an R-node and an I-node.

It is possible, in theory, to differentiate implication semantics from the conventional implementation coupling already included in the model between R-nodes and I-nodes. Nevertheless, it is unlikely that there would be a real need for a capability to represent such implication information, for two reasons. First, presumably all implications would have the same "orientation": from R-node to I-node. After all, the requirement nodes are the starting point for the whole process, and

implementation issues normally follow from requirements. Secondly, it is unlikely that a designer would perceive a relationship between an R-node and an I-node that would not correspond to an implementation link, and hence already be included in that form. Of course, the implementation link would represent somewhat different information, but the fact that they join the same nodes means that the implication link would probably have little additional design structuring impact. For these reasons, it is concluded that there is probably little additional R-to-I-node (or vice versa) semantic information that a designer would want, or be able, to include in the design model, that is not captured already in the form of conventional implementation links. Therefore, the alternative of logical implication between R-node and I-node will not be included in the extended design structuring model.

## 4. Application of Extended Model to the 22-node DBMS Requirements Set

In order to illustrate an application of the extended architectural design model described in this paper, a set of 22 requirements for a data base management system (DBMS), studied earlier by Andreu (see (Andreu, 78)), are analyzed here. The 22 requirements, given in Section 4.1 below, are only skeleton requirements for a real DBMS, but they are quite satisfactory for demonstration purposes. In a later report, the extended model will be applied to a larger, more realistic requirements set.

In making the assessments reported here, the following steps were followed.

(1) The requirements as used by Andreu were adopted in their entirety. The requirement statements were then transformed into "template form" (see (Huff 78) and the Appendix for more information regarding requirement statement templates).

(2) The interdependency assessments made by Andreu in earlier analysis of these requirements were also used here. While this author (and, for present purposes, DBMS designer) does not necessarily agree completely with the appropriateness of the interdependency assessments reported by Andreu, no significant changes were made so as to maintain comparability with the earlier results. The descriptions of these interdependencies, given in Section 4.2, were clarified and expanded from their original form.

(3) Weights were assigned to each interdependency. The weight interpretation used here was that of strength of interaction, as described in Section 3.1. Assigned weights were either W (weak), A (average), or S (strong).

(4) Interdependencies were reviewed to determine similarities. This review was conducted as follows: first, an interdependency between, say, requirements n and m was selected. Then, the concept underlying the description of this interdependency was compared, mentally, to the underlying concepts for all other interdependencies in which either requirement n or m was involved. Similarity assessments were rated W, A, or S, similar to interdependency assessments.

(5) Requirements were again reviewed pairwise to search for implication relationships, both lateral and hierarchical. Because this experimental requirements' set is quite small, each requirement statement is rather highly abstracted. Hence the liklihood of many logical interactions is small. In fact, only three such relationships were identified.

(6) Finally, interdependencies were reviewed pairwise to determine logical relationships. For the same reason as given in (5) above, few such relationships were expected (only one was identified).

The results of these analyses are reported in the following five sections. Section 4.1 lists the original DBMS requirements (in template form); 4.2 describes the assessed interdependencies and the weights assigned; 4.3 gives the interdependency commonality assessments and their assigned weights; 4.4 describes implication relationships between requirements; and, 4.5 gives the implication relationships between interdependencies.

Figure 4.1 is a schematic representation of the extended model as applied to the 22-node DBMS design problem, using the diagrammatic techniques discussed in Section 3. Section 4.6 includes comments on the process of making the various kinds of assessments demanded by the extended model.

4.1 Sample Set of 22 DBMS Requirements.

The requirements given below are modified versions of the 22 requirements analyzed in (Andreu, 78). For a brief explanation of the nature of, and the motivation behind, the modifications, refer to the Appendix.

SAMPLE SET OF 22 DBMS REQUIREMENTS.

1. The database can have multiple logical organizations.
2. There can be user-meaningful logical data-item groups.
3. There can be user-meaningful relationships among data items.
4. There can be algorithmic relationships among data items.
5. There will be logical operations involving data items, groups, and relationships.
6. Data items will be organized physically in a unique way.
7. There will be certain specific queries.
8. Frequencies of queries will be non-uniform.
9. Data items may be referenced according to logical group membership.
10. Data items may be referenced according to item value.
11. Data item retrieval will be as fast as possible.
12. The distribution of data items across queries can be significantly non-uniform.
13. There will be an English-like language for expressing queries.
14. The query language will be unambiguous.
15. Query expressions will be non-procedural.
16. There will be different data item types (e.g., integers, character strings).
17. Same type data items can be combined using certain operations

(e.g., addition, for integers; concatenation, for character strings).

18. Certain data items may be represented using alternative data types.

19. There will be a specific value range for each data item.

20. Data items can take values from a subset of their value range.

21. Data redundancy will be minimized.

22. Storage costs will be minimized.

4.2 Requirements' Interdependencies and Weights.

The assessed interdependencies and assigned weights for the 22 DBMS requirements are given below. The numbers in the "requirement pair" column refer to requirement statements from the previous section. Weight codes are:

W - weak

A - average

S - strong.

INTERDEPENDENCIES BETWEEN DBMS REQUIREMENTS

| Requirement Pair | Weight | Interdependency Description |
|---|---|---|
| (1,2) | A | Logical views are definable in terms of logical groups |
| (1,3) | A | Logical views are definable in terms of relationships among data items. |
| (1,5) | A | Logical operations are carried out in the context of the logical view. |
| (1,6) | S | Various logical views must be obtainable from a unique physical view. |
| (1,21) | A | Data redundancy would otherwise be useful in implementing multiple logical views. |
| (2,3) | W | Relationships may exist both within and between logical groups. |
| (2,5) | A | Logical groups may participate in logical operations. |
| (3,5) | S | Logical relationships may be involved in logical operations. |

| | | |
|---|---|---|
| (4,17) | A | Algorithmic relationships among data items must be consistent with allowable operations. |
| (4,18) | A | Algorithmic relationships must be defined in terms of allowable alternative data types. |
| (4,21) | A | Depending on how implemented, algorithmic relationships can help to avoid redundancy. |
| (4,22) | S | By virtualizing certain data, storage requirements may be reduced through use of algorithmic relationships. |
| (5,7) | S | Queries must be compatable using defined operations. |
| (5,15) | A | Mapping(s) must exist between non-procedural queries and the set of logical operations. |
| (6,9) | W | Logical group membership should unambiguously correspond to membership in some part of the unique physical organization. |
| (6,21) | W | A unique physical organization favors non-redundancy. |
| (7,13) | W | All queries should be expressible in the English-like query language. |
| (7,14) | W | All queries should be unambiguous. |
| (7,15) | W | All queries should be expressi-le in non-procedural fashion. |
| (8,11) | S | Data physical organization should reflect query type frequency to minimize lookup time. |
| (8,12) | A | Frequencies of queries and frequencies of data items within queries determine frequencies of data items' references. |
| (9,11) | S | Data item location via logical group membership can affect query response time. |
| (9,12) | W | Database searching mechanisms should take into account |

| | | |
|---|---|---|
| | | distribution of data items in logical groups. |
| (9,21) | A | Representing every logical group physically is an alternative that goes against avoiding redundancy. |
| (10,11) | V | Alternative mechanisms for locating data items by value have efficiency implications. |
| (10,12) | A | Alternative mechanisms for locating data items by value should take into account their frequency of reference in queries. |
| (10,19) | A | Bounds checking can be used to resolve references by data type. |
| (10,20) | W | A data reference value may fall within the relevant range for that data item, yet not be in the data base. |
| (11,19) | A | Bounds checking can enhance response time for certain queries. |
| (11,20) | W | Knowledge of relevant target values may help in choice of most efficient search strategy. |
| (11,22) | S | More efficient lookup strategies usually require more memory. |
| (13,14) | W | The more English-like a query language, the greater the scope for ambiguous queries. |
| (14,15) | W | The less procedural the query language, the greater the opportunity for ambiguity. |
| (16,17) | A | Operations must be consistent with the type of data item upon which they act. |
| (16,18) | A | as for (16,17) |
| (16,22) | W | Certain data types may be stored more compactly than others. |
| (17,18) | S | Operations must al5ays be consistent with the data types used. |

| | | |
|---|---|---|
| (18,22) | A | Certain data types may be stored more compactly then others. |
| (21,22) | A | Decreased redundancy means decreased storage costs. |

4.3 Interdependency Similarity Assessment.

Assessed similarity relationships and weights between pairs of interdependencies are given below. The numbering scheme for interdependency pairs refers to the original requirements' numbers. For example, (1,2) corresponds to the interdependency between requirements 1 and 2. The weight codes used are the same as used for interdependency weights (see Section 4.2).

INTERDEPENDENCY SIMILARITY ASSESSMENTS

| Interdependency Pair | Weight | Description of Nature of Similarity |
|---|---|---|
| (1,2),(1,3) | A | Common issue regarding definition of logical groups. |
| (2,5),(3,5) | A | Participation in logical operations. |
| (4,17),(4,18) | S | Common virtualization issue. |
| (4,21),(4,22) | A | Common virtualization issue. |
| (10,11),(10,12) | A | Similar efficiency issues. |
| (10,11),(10,19) | A | Both related to value representation. |
| (10,19),(10,20) | A | Both related to data reference by value. |
| (13,14),(14,15) | S | Both concern query language design. |
| (16,17),(16,18) | S | Same issue. |
| (16,22),(18,22) | S | Both related to data type selection issue. |

## 4.4 Implication Relationships Between Requirements.

Three implication relationships between requirements were determined, shown below. The first two are one-way lateral implications, the third is a two-way lateral implication. No hierarchical implication relationships were detected in the 22-node DBMS set.

IMPLICATION RELATIONSHIPS BETWEEN REQUIREMENTS

| Requirement Pair | Nature of Relationship | Comments |
|---|---|---|
| 6,21 | 21 --> 6 | To minimize data redundancy, the system ought to support a unique physical organization of data. |
| 16,18 | 18 --> 16 | In order to be able to represent certain items using various data types, the system must support an appropriate variety of data types. |
| 21,22 | 21 --> 22 | Reducing data redundancy tends to reduce storage costs. |

4.5 Implication Relationships Between Implementation Issues.

A single one-way implication relationship between interdependencies was assessed, as given below.

IMPLICATION RELATIONSHIPS BETWEEN INTERDEPENDENCIES

| Interdependency Pair | Nature of Relationship | Comments |
|---|---|---|
| (10,11),(10,12) | (10,12) --> (10,11) | An implementation of a mechanism for locating data items by value that takes into account the items' distribution across queries will generally also be more efficient in terms of retrieval response. |

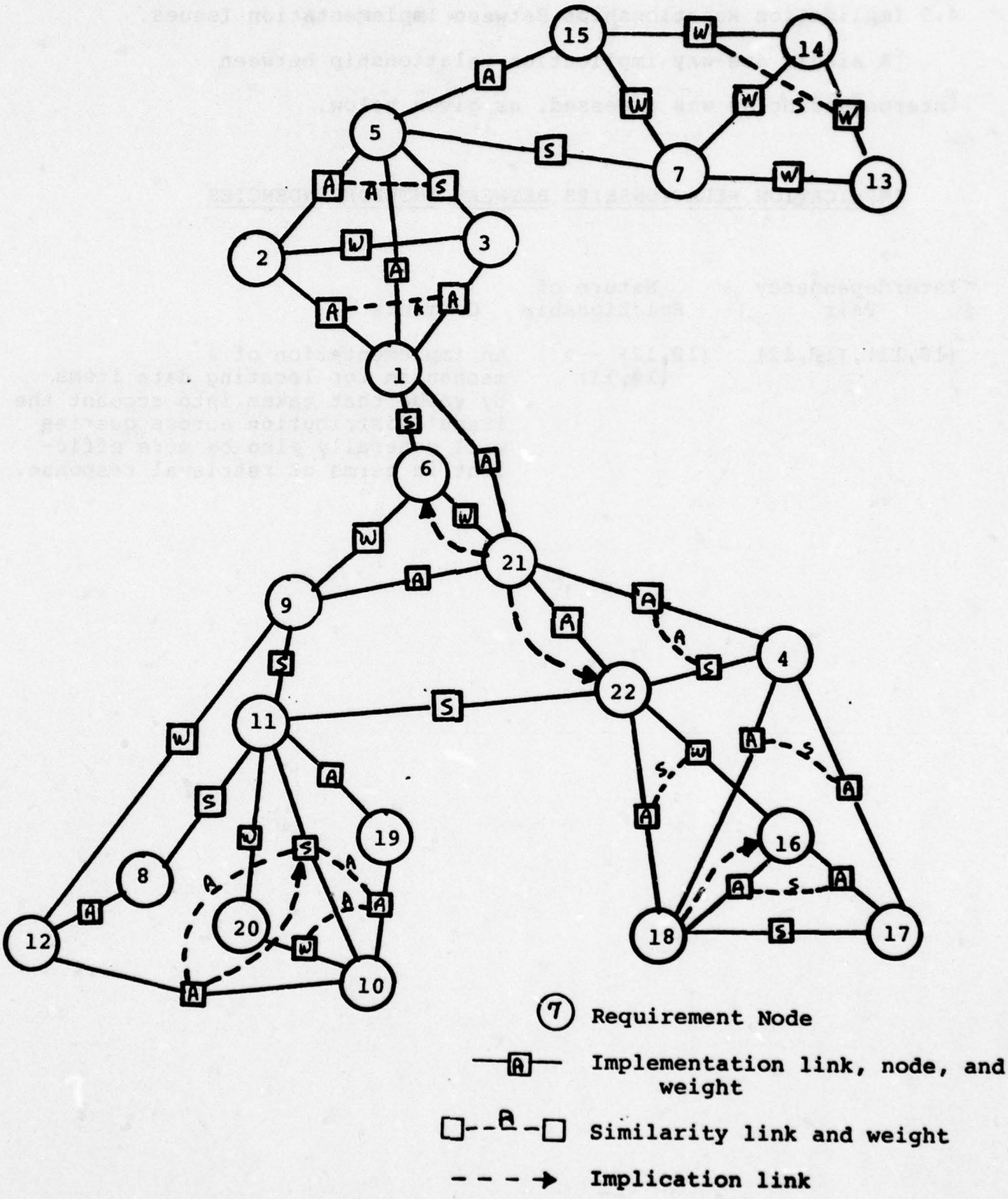



Figure 4.1

Graph Diagram of DBMS Design Problem Using Extended Model.

4.6 Comments on Assessments.

In reporting his experiences applying the basic model, Andreu commented on two aspects of assessing requirement interdependencies. First, he pointed out that, although the number of actual comparisons between pairs of requirements increases as the square of the number of requirements, the actual designer effort involved in making the assessments is not nearly so large as this would seem to imply. The primary reason is that most of the requirements have no significant interdependencies, and may be assessed quite rapidly. In working through an experimental design consisting of over 100 requirement statements, Andreu found over 93 percent of the requirement pairs were of this "easy" variety.

The second point Andreu made was that it turned out to be relatively easy to conjure up "mental models", or implementation schemes, for pairs of requirements:

> "From a personal experiential viewpoint, we must say that the models emerged rather naturally from confronting pairs of requirements, and more easily than expected." (1)

In the course of the analysis performed on the 22-node DBMS requirement set in this report, Andreu's two points were found to hold. Also, experience gained in the determination of the additional kinds of information - interdependency weights, interdependency similarities and associated weights, and

---

(1) see (Andreu 78, page 234)

implication relationships - is summarized as follows:

(1) Weight Assessments.

It was earlier suggested by Andreu that it might be possible to make weight assessments by counting the number of different implementation schemes that underly a given interdependency, then assigning a weight value based on the total number of such schemes. This approach was not found to be very useful, for two reasons.

First, most of the related requirement pairs were actually related via a single interdependency. However, in a number of cases, the strengths of these single interdependencies were judged significantly different, hence deserved different weight values. Andreu's proposed approach would not properly handle this situation.

The other main reason is that many of the conceptual models of interdependence were sufficiently general in nature that it wouldn't be meaningful to try to "count" them as individual implementation schemes.

Rather than attempt to use a mechanical approach to determining the weights to be assigned to each interdependency, the assessment was made judgmentally, i.e., by mentally examining the same conceptual models used in determining the interdependencies themselves in the first place. It must be granted that such a judgmental approach to eliciting the strengths of interdependencies would probably lead to a fairly high variability among different

designers in practice. However, this variability could be reduced somewhat, possibly using a Delphi-like technique which would have the designer re-think his assessments in light of assessments made by other designers.(2) Also, some variability among different designers is to be expected, inasmuch as the different designers perceive the many design issues in different ways.

The above comments also apply to assessment of the strengths of similarity relationships between implementation considerations.

(2) Implication Relationship Assessments.

Few implication relationships were detected in the set of 22 DBMS requirements. The main reason for this is that the requirement statements were few in number and broad in scope. It is expected that a more realistic (i.e., larger) requirements set would exhibit a higher proportion of requirement and implementation issue implication relationships.

Those relationships that were assessed in the 22-node requirement set were actually identified rather easily. The scanning of requirements statements or interdependency description statements to locate implication relationships

---

(2) There are many similarities between the Delphi technique used in forecasting and "opinion averaging" analysis, and some of the modern programming management methods such as "structured walkthroughs".

could be done rapidly, and related statements generally stood out quite plainly.

On balance, the additional assessment time and effort needed to determine the extra information required by the extended model was somewhat less than that needed to determine the original basic interdependencies. The demands that would be placed on a designer to supply this data are non-trivial, but not out of the realm of reason, assuming the designer believes that the final quality of the design structure will be significantly improved as a result.

## 5. Summary.

The argument in this paper has been that there are important additional kinds of information designers generally possess that cannot be represented in the basic architectural design framework developed originally by Andreu. Certain classes of such untapped information, believed to be most relevant and accessable via designer judgment and knowledge, have been identified. Possible schemes for representing much of this additional information in the context of an extended graph model have been discussed.

To summarize the various kinds of information proposed for the extended architectural design framework, consider Figure 5.1. Shown here is the schematic representation of certain design information, using the basic design model, involving five functional requirements (1,2,3,4, and 5). A designer, having studied the 10 requirement pairs, has identified five implementation interrelationships (1-2, 1-3, 2-3, 1-4, and 1-5).

Adding extension 1 to the representation - weights on the implementation links - the designer's assessments might be as shown in Figure 5.2.

Extension 2 is then added: the implementation issues are explicitly represented as I-nodes, and similarities among them are determined and added to the diagram. Assuming a single such similarity relationship, suppose implementation nodes (1-2) and (1-3) are determined to represent similar issues (within estimated 50 percent average overlap). This information is portrayed in Figure 5.3.

Finally, additional semantic information, in the form of

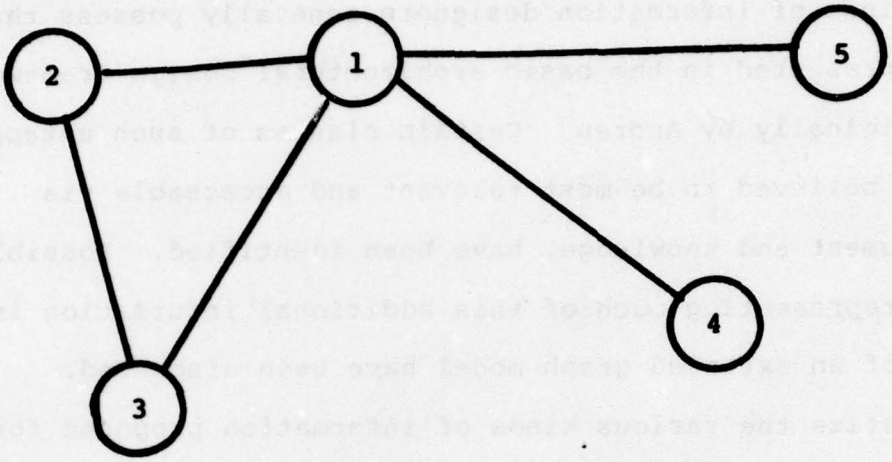



Figure 5.1

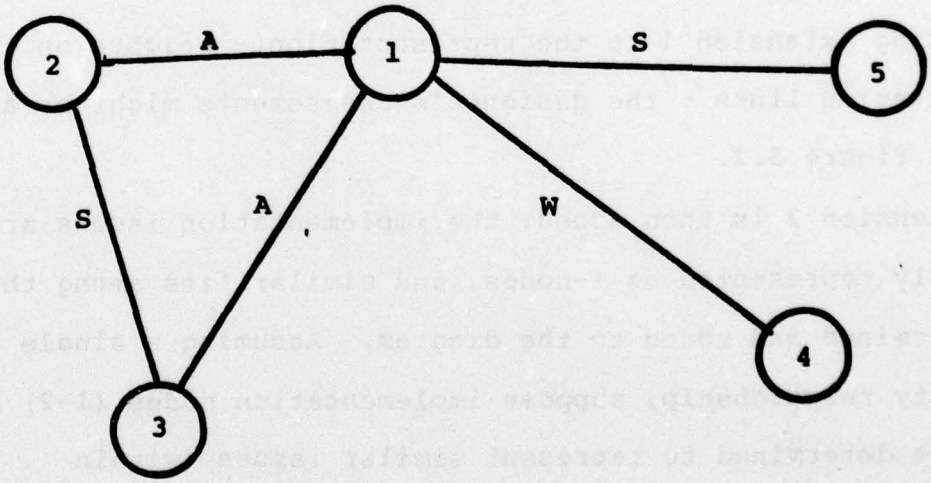



Figure 5.2

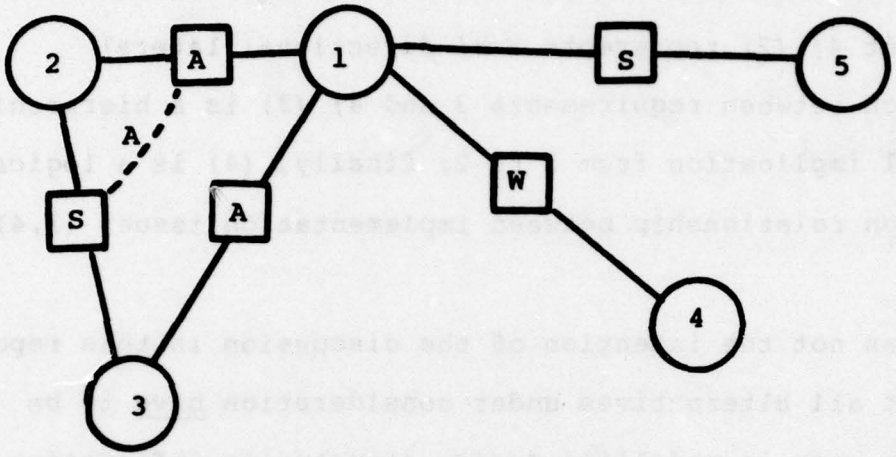



Figure 5.3

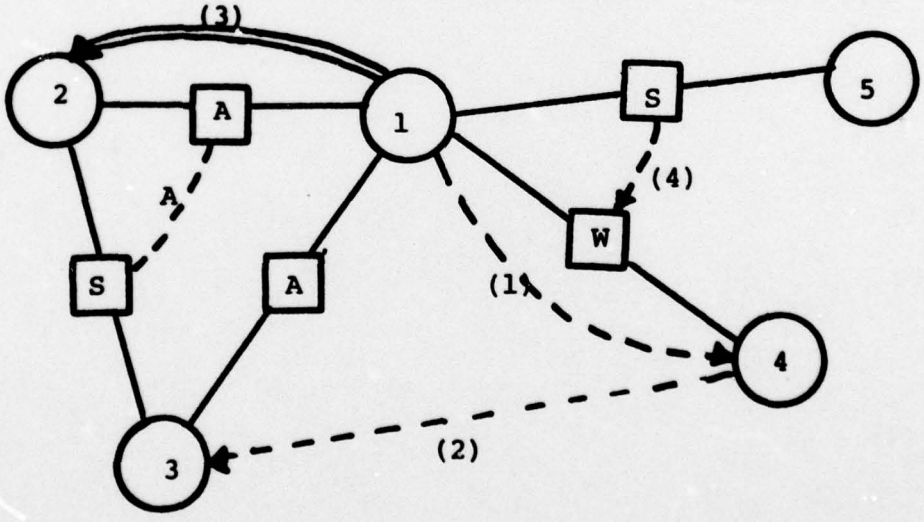



Figure 5.4

logical implications at both functional level and implementation level are assessed and added to the schematic. In Figure 5.4, (1) represents a lateral implication from requirement 1 to requirement 4; (2) represents a bi-directional lateral implication between requirements 3 and 4; (3) is a hierarchical functional implication from 1 to 2; finally, (4) is a logical implication relationship between implementation issues (1,4) and (1,5).

It was not the intention of the discussion in this report to imply that all alternatives under consideration <u>have</u> to be brought to bear in modelling design structuring information - only that they <u>might</u> be useful. Specifically, ways in which such information could be incorporated into design structuring algorithms will be the subject of a later report.

## REFERENCES

1. Alexander, C.: Notes on the Synthesis of Form, Harvard University Press, 1964.

2. Andreu, R. C.: "A Systematic Approach to the Design and Structuring of Complex Software Systems", PhD Thesis, MIT Sloan School of Management, 1978.

3. Holden, T.: "A Systematic Approach to Designing Complex Systems: An Application to Software Operating Systems", MIT Center for Information Systems Research, Technical Report #5, NTIS No. AD-A055481, May 1978.

4. Huff, S. L.: "An Approach to Constructing Functional Requirement Statements for Preliminary Systems Design", MIT Center for Information Systems Research, Technical Report #6, June 1978.

5. Madnick, S. E., and R. C. Andreu: "Research on a Systematic Approach to the Design of Complex Systems: Application to Data Base Management Systems", Research Proposal submitted to the Naval Electronic Systems Command, November 1977.

# APPENDIX

## The 22-node DBMS Requirements: Comparison of Original Statements and Template Form

Functional requirement statements are usually expressed in standard English descriptive prose. However, in order to apply the graph representation and decomposition techniques developed in the course of this research, it is necessary to express requirements as short (one-sentence) statements having the properties outlined in Section 2.1 earlier. One approach to transforming prose-form requirement statements into this unifunctional statement form has been investigated and reported in (Huff 78). That scheme involves the use of a set of six requirement statement "templates", or general statement patterns, which can be used to map prose specifications into a relatively standardized set of statements suitable for further analysis using the graph partitioning methodology.

In order to better couple this report to the earlier one, and to illustrate the applicability of the template scheme, each of the 22 DBMS-requirements is given in Appendix A.2 below, expressed first as originally stated by Andreu, then in template form. Appendix A.1 first defines the six statement templates used to restructure the DBMS requirements.

## A.1 Functional Requirement Specification Templates.

The six types of requirement specification templates, originally identified and discussed in (Huff 78), are given below:

A. Existence.

There (can/will) be <modifier> <object>

B. Property.

<Mod> <object> (can/will) be <mod> <property>

C. Treatment.

<Mod> <object> (can/will) be <mod> <treatment>

D. Timing.

<Mod> <object> (can/will) <timing relationship> <mod> <object>

E. Volume.

<Mod> <object> (can/will) be <order statement> <index> <count>

F. Relationship.

a. Subsetting.

<Mod> <object> (can/will) contain <mod> <object>

b. Independence.

<Mod> <object> (can/will) be independent of <mod> <object>

A.2 Transformation of the 22 DBMS Requirements to Template Form.

Each of the 22 DBMS requirements defined originally by Andreu, and used in this and earlier reports, was transformed to one of the six template forms. Given below are each of the original (as per (Andreu 78)) statements, followed by the revised statement in template form. The revised statements are the same as those given in Section 4.1 earlier. The template type used to re-write each statement is also noted.

Two points are worth noting here. First, it turned out that only three of the six templates - Existence, Property, and Treatment - were actually used in performing these translations. The main reason for this is simply that this requirements' set is a small, experimental one. (All six template forms were used in transforming a more realistic, larger set of DBMS requirements, as reported in (Huff 78)). On the other hand, as a practical matter, the absence of instances of certain templates might suggest that some classes of requirements for the target system had not been adequately specified. Recognition of this possibility should lead to a better, more complete functional specification.

## ORIGINAL AND TRANSFORMED REQUIREMENTS STATEMENTS

1. Original: The collection of data items that is to be supported should be perceivable as logically organized in more than one way.

   Transformed (Existence): The database can have multiple logical organizations.

2. Original: Data items should be perceivable as forming logical

groups, meaningful to the user.

Transformed (Existence): There can be user-meaningful logical groups of data items.

3. Original: Relationships exist among data items, meaningful to the user.

   Transformed (Existence): There can be user-meaningful relationships among data items.

4. Original: Algorithmic relationships among data items should be supported.

   Transformed(Existence): There can be algorithmic relationships among data items.

5. Original: There is a collection of logical operations involving data items, groups, and relationships (the data manipulations) that must be supported.

   Transformed (Existence): There will be logical operations involving data items, groups, and relationships.

6. Original: Data items are to be organized physically in a unique way.

   Transformed (Treatment): Data items will be organized physically in a unique way.

7. Original: There is a number of specific queries to be supported.

   Transformed (Existence): There will be certain specific queries.

8. Original: Query frequency is not uniform - i.e., there are certain critical queries.

   Transformed (Property): Frequency of queries can be non-uniform.

9. Original: In a query, references to data items may be made by logical group membership.

   Transformed (Treatment): Data items may be referenced according to logical group membership.

10. Original: In a query, references to data items may be made by item value.

    Transformed (Treatment): Data items may be referenced according to item value.

11. Original: The expected time spent in locating thee data items appearing in a given query should be minimized.

    Transformed (Property): Data item retrieval time will be as short as possible.

12. Original: The distribution of data items across queries (data items appearing in a query) is far from uniform, in general.

    Transformed (Property): The distribution of data items across queries can be significantly non-uniform.

13. Original: Queries are to be expressed in an English-like language.

    Transformed (Existence): There will be an English-like language for expressing queries.

14. Original: The query language should be unambiguous.

    Transformed (Property): The query language will be unambiguous.

15. Original: Query expressions should be non-procedural.

    Transformed (Property): Query expressions will be non-procedural.

16. Original: Different types of data items must be supported (e.g., integers, character strings).

    Transformed (Existence): There will be different data item types (e.g., integers, character strings).

17. Original: Data items of the same type can be combined by means of certain well-defined operations (e.g., addition, for integers; concatenation, for character strings).

    Transformed (Treatment): Same type data items can be combined using certain operations (e.g., addition, for integers; concatenation, for character strings).

18. Original: Alternative data types may be employed, if necessary, for certain data items.

    Transformed (Treatment): Certain data items may be represented using alternative data types.

19. Original: Each data item takes values in a specific range of its data type.

    Transformed (Existence): There will be a specific value range for each data item.

20. Original: Data items do not necessarily take on all values in their value range.

    Transformed (Property): Data items can take values from a subset of their value range.

21. Original: Data redundancy should be avoided.

    Transformed (Treatment): Data redundancy will be minimized.

22. Original: Storage costs should be minimized.

    Transformed (Treatment): Storage costs will be minimized.